ASIP design based on CORDIC algorithm using Xilinx and CoWare designer tools

A Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

by

Debabrat Mishra (107EC013)

under the guidance of

Prof. Kamala Kanta Mahapatra

Professor

Department of Electronics and Communication, NIT Rourkela

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

ROURKELA

2011

ASIP design based on CORDIC algorithm using Xilinx and CoWare designer tools

A Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

by

Debabrat Mishra (**107EC013**)

Department of Electronics and Communication, NIT Rourkela

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

ROURKELA

2011

National Institute of Technology
Rourkela

CERTIFICATE

This is to certify that the thesis entitled, **"ASIP design based on CORDIC algorithm using Xilinx and CoWare designer tools"** submitted by **Debabrat Mishra(107EC013)** in partial fulfillment of the requirements for the award of **Bachelor of Technology Degree** in **Mechanical Engineering** at National Institute of Technology, Rourkela is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in this thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Date:

Prof. K.K.Mahapatra
Dept. of Electronics and Comm. Engineering
National Institute of Technology
Rourkela-769008

ACKNOWLEDGEMENT

I am extremely grateful to my guide Prof. K.K. Mahapatra for giving me the opportunity to work under him and for giving the topic to work on, which was both extremely interesting as well as challenging. His timely advice from time to time, however, made life much easier than it would have been.

I am also grateful to Mr. Jagannath, Mr Anup Sarma and Mr Soubhagya Sutar without whose help, CoWare would have always remained an unsurmountable task. It was because of their help and tips that I was able to complete my thesis on time.

Last but not the least, I am thankful to my family, all my professors and friends without whose support this work would never have been possible.

Date:

Debabrat Mishra
(107EC013)
Dept. of Electronics and Comm. Engineering
National Institute of Technology
Rourkela-769008

ABSTRACT

Efficient generation of trigonometric as well as exponential functions without much increase in hardware complexity has always been a challenge, owing mainly to their importance and widespread use in Digital Signal Processing applications besides other areas. One such algorithm which is very much effective for the calculation of trigonometric, hyperbolic, exponential, linear and logarithmic functions is the CORDIC algorithm. The algorithm is very much hardware efficient because it omits the dependence on multipliers and is rather a combination of shift-add operations.

Application Specific Instruction-set Processors (ASIPs) are a type of processor that serve as a compromise between General Purpose Processors (GPPs) and Single Purpose Processors (SPP). Their data-path can be optimized for a particular class of operations such as embedded control, Digital Signal Processing (DSP) applications etc. This project deals with the design of an ASIP based on the CORDIC algorithm using two very popular hardware designing tools, i.e , Xilinx Integrated Development Environment (IDE) from Xilinx corporations, Inc. and LISA 2.0 description language and processor designing environment from CoWare.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1 The CORDIC algorithm

The **CO**-ordinate **R**otation **DI**gital **C**omputer (CORDIC) [1] is a special purpose computer meant for the real-time calculation of trigonometric and exponential functions by the use of iterative vector rotations. Vector rotations can also be used for the conversion of polar to rectangular and polar to rectangular coordinate conversions. The algorithm can be derived from the rotation transform :

$$x' = x.\cos \phi - y.\sin \phi$$

$$y' = y.\cos \phi + x.\sin \phi$$

On rearrangement of the terms, this can be given as :

$$x' = \cos \phi \, [\, x - y.\tan \phi \,]$$

$$y' = \cos \phi \, [\, y + x.\tan \phi \,]$$

The implementation of these equations is still complex due to the presence of the trigonometric functions. However, if the rotation angles are restricted to values such that $\tan \phi = \pm 2^{-i}$, the multiplication by the tangent can be greatly simplified as it can be implemented using simple shift operations . Thus, arbitrary angles can be obtained by performing a series of rotations iteratively. At each rotation, the direction of rotation is chosen by obtaining the difference between the actual angle and the angle obtained by rotation. Mathematically, it can be given as shown in the next page.

$$x_{i+1} = K_i [x_i - y_i . d_i . 2^{-i}] \quad (1)$$

$$y_{i+1} = K_i [y_i + x_i . d_i . 2^{-i}] \quad (2)$$

where , $K_i = \cos (\tan^{-1} 2^{-i}) = 1/ \sqrt{} (1 + 2^{-2i})$

$d_i = \pm 1$

The value of d_i is +1 if the angle to be obtained is greater than the current iterative angle and is - 1 if the current iterative angle is greater. The value of K_i can be taken to be a constant with a value of about 0.6073 when the number of iterations is taken large.

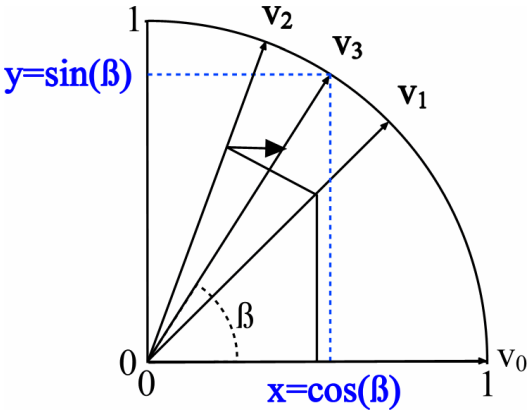


Fig.1. Vector rotation method of sine and cosine calculation

While computing we start with an initial value of x-coordinate at 1 and an initial value of y-coordinate as 0. In the first iteration, the vector rotates by an angle of 45°, which gives us the first iteration result. If this angle is greater than the angle β, the next rotation takes place in the reverse direction, else in the same direction. Finally, after the specified number of rotations, the value of cos(β) is given by the x-coordinate while the value of sin(β) is given by the y-coordinate.

1.2 Types of CORDIC algorithm [2]

CORDIC algorithm for the calculation of sine and cosine values is of three types. Each of the types have their own advantages and disadvantages.

The three types are :

1. Sequential / iterative
2. Parallel / cascaded
3. Pipelined

1.2.1 Sequential / iterative CORDIC :

In this type of CORDIC, a single iteration takes place in one clock cycle. The basic hardware structure of sequential CORDIC algorithm is as shown :

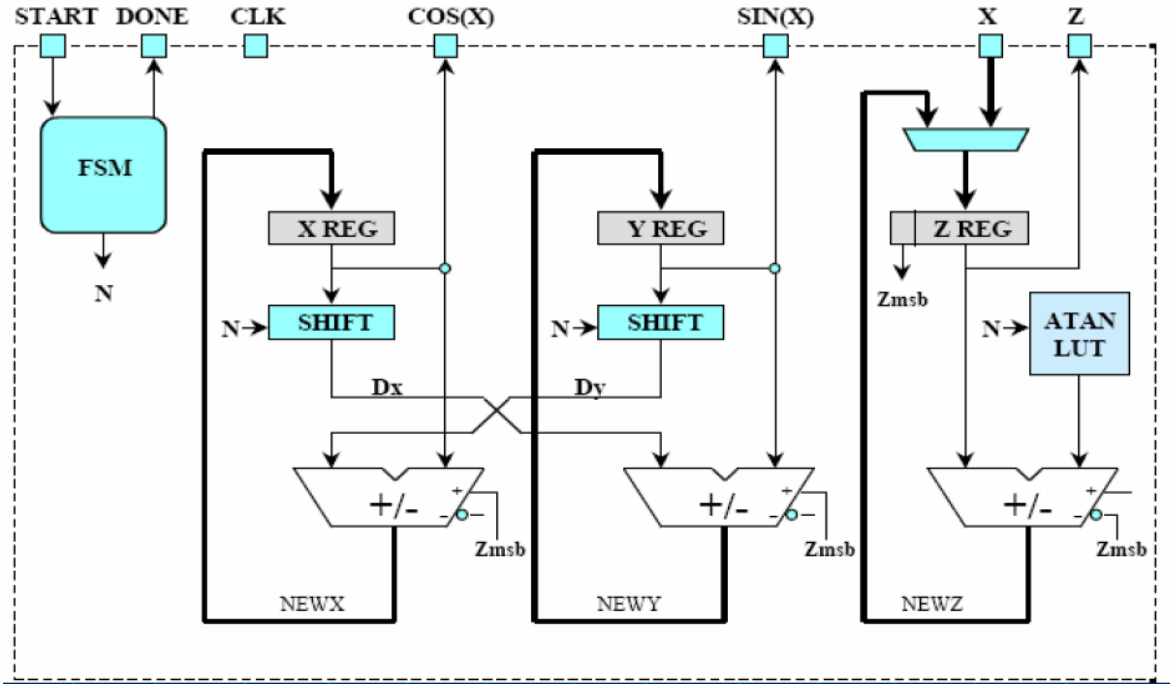


Fig.2. Sequential / iterative CORDIC structure [2]

Advantages :

1. The hardware complexity is least and it occupies the least area.
2. It has maximum number of clock cycles per iteration.
3. Power consumption is least.

Disadvantage :

1. Maximum number of clock cycles are required to calculate the output, thus calculation time is very slow.
2. Variable shifters do not map well on certain FPGAs due to high fan-in.

1.2.2 Parallel / cascaded CORDIC :

In this type of CORDIC algorithm, all the calculations take place within a single clock cycle. The hardware of parallel CORDIC algorithm is as shown :

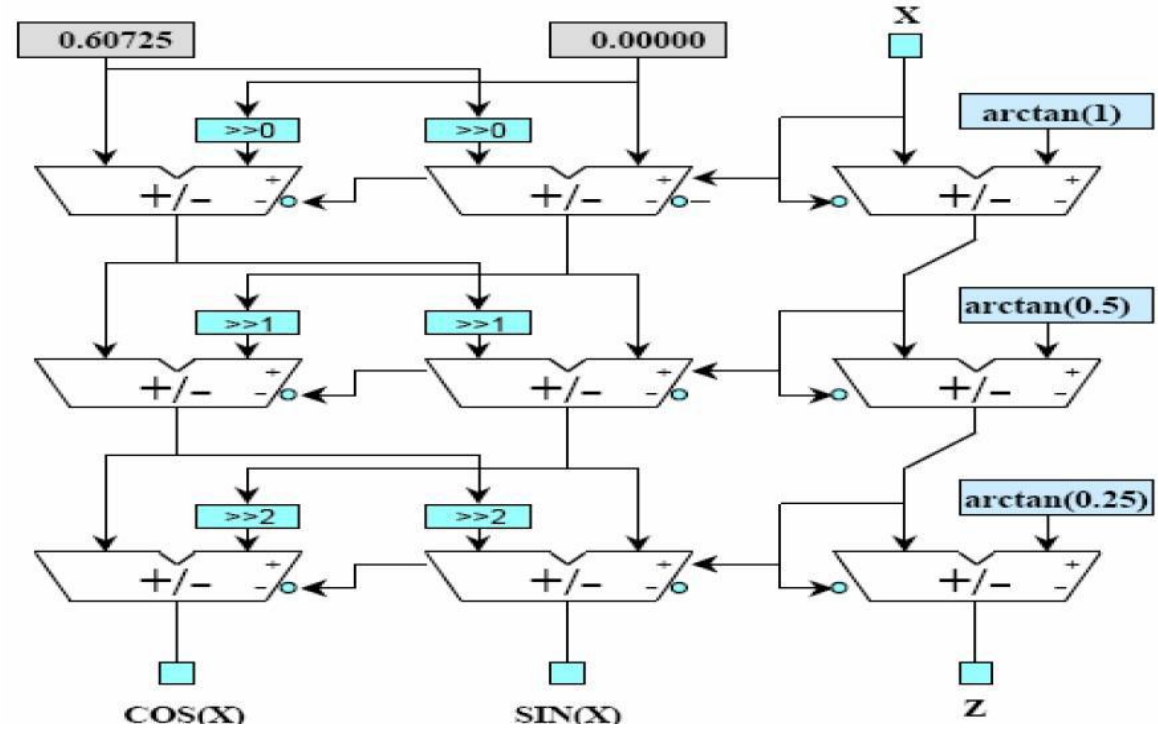


Fig.3. Parallel / cascaded CORDIC structure [2]

Advantages :

1. It has considerable delay, but processing time is reduced as compared to the iterative process.
2. Shifters are of fixed size and so can be implemented in the wiring.
3. Constants can be hardwired instead of requiring storage space.

Disadvantages :

1. The amount of hardware required is large and the area required is maximum
2. Power consumption is highest among the three CORDIC architectures.

1.2.3 Pipelined CORDIC :

It is the most efficient of the CORDIC algorithms in which the iterations take place in multiple clock cycles. However, different processes take place concurrently such that the execution time is reduced. The structure of pipelined architecture is as shown :

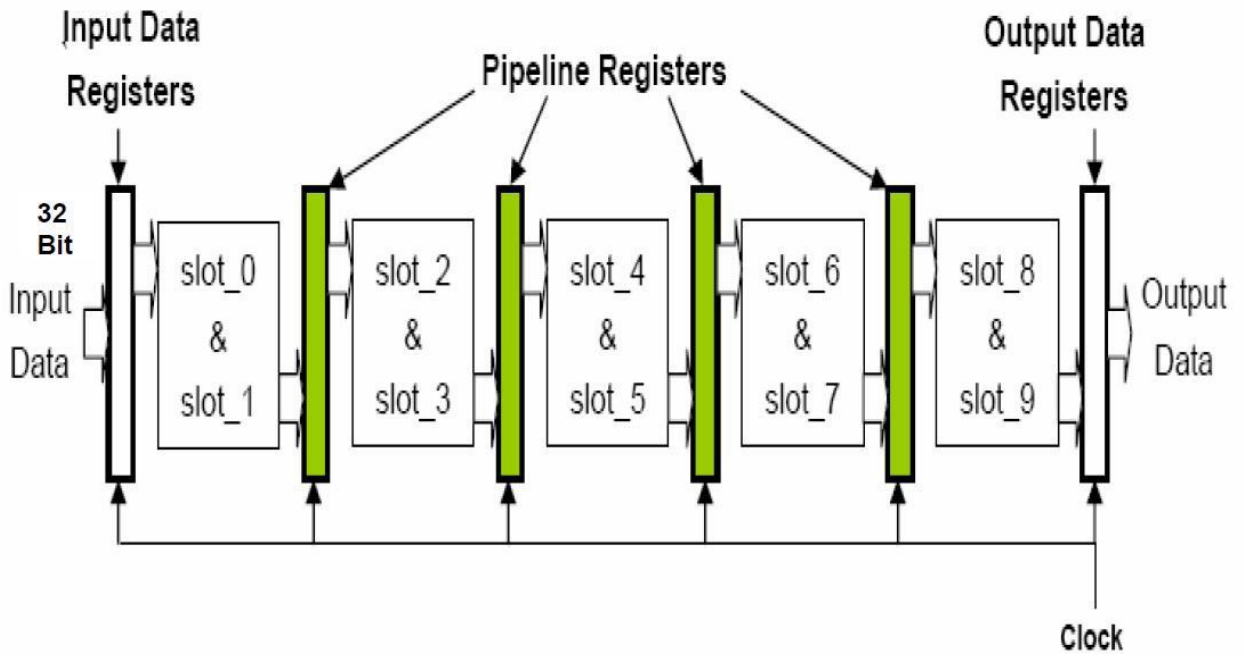


Fig.4. Pipelined CORDIC structure [2]

Advantages :

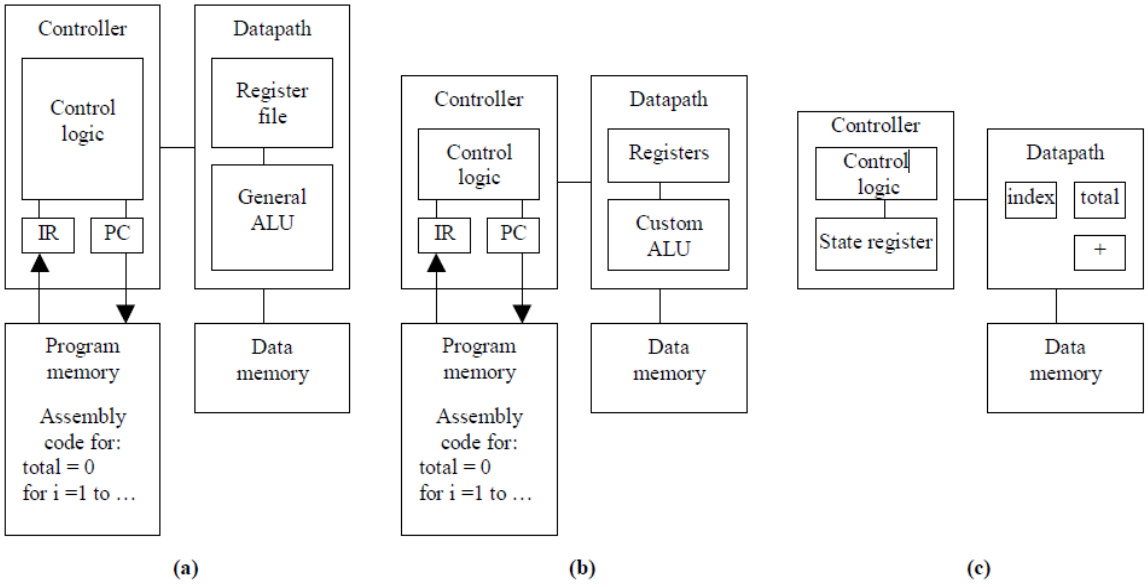
1. FPGA implementation is easy, as registers are already available, thus requiring no extra hardware.
2. Number of iterations after which the system gives accurate result can be modeled, considering clock frequency of the system.
3. When operating at greater clock period power consumption in later stages reduces due to lesser switching activity in each clock period.

Disadvantages :

1. Hardware complexity as well as area required is more than sequential architecture.
2. Power consumption is lower than parallel but higher than sequential structure.

1.3 Types of processors [3]

Processors mainly refer to the architecture of the computation mechanism employed to obtain the desired functionality of a system. The processors may be programmable or non-programmable, depending upon the application. They can be specialized and implement only a single function, or be general purpose and implement a wide range of functions. The main feature which governs the use of different types of processors for different applications are the design metrics. Some of the most commonly considered design metrics are NRE cost, flexibility, performance, power consumption, size, time-to-prototype and so on. The different types of processors are as shown in the next page.



[3]

Fig.5. (a) General Purpose,(b)Application-specific & (c)Single Purpose Processors

1.3.1 General Purpose Processors

A General Purpose Processor(GPP) or microprocessor as it is generally called, is a programmable device that has the aim of implementing a large number of applications such that the number of devices sold is maximized. The main features of this processor are that , the program memory is not built-in to the circuit, since it has to run different programs at different times and it has a general datapath , with a large register file and one or more general purpose Arithmetic and Logical Units (ALUs).

It has good time-to-market and NRE costs since only the program has to be changed for the different applications without any change in hardware. Flexibility is also high due to the same reason. However, the performance is poor for certain applications and the size and power consumed are also high, because of the large hardware size.

1.3.2 Single Purpose Processors

A Single Purpose Processor(SPP) is a processor or a digital circuit which is designed to execute only a single program. For example, the circuit used for image processing in a digital camera is a SPP which has the single function of processing the input image and storing it for subsequent retrievation. It has almost the opposite features of a GPP, since it has a small register file, a dedicated datapath with an ALU performing only a limited number of operations and no provision of altering the program memory.

It has several design benefits, since the performance may be fast, power consumption less and also small size. However, it has the disadvantages of having very high NRE costs, low flexibility and longer design time.

1.3.3 Application Specific Instruction-set Processors

An Application Specific Instruction-set Processor (ASIP) serves as a compromise between a GPP and a SPP. It is a programmable processor which has an optimized datapath for implementing only a particular class of operations. Several special functionalities may be added while unnecessary ones eliminated. Microcontrollers and Digital Signal Processors (DSPs) are some of the most common types of ASIPs in use. They have a program memory that can be changed for different applications and limited register-memory file depending upon the type of application and memory use.

It has the advantages of having flexibility, at the same time achieving good performance, low power consumption and optimum size. The drawback is that it requires large NRE cost to manufacture, especially to design the compiler. Certain design environments such

as CoWare offer the benefit of automatically generating the compiler which has greatly reduced the cost and time of manufacturing the device.

1.4 Organisation of thesis

Second chapter deals with the literature survey on the different variations of the cordic algorithm and its implementation on different platforms and design environments.

Third chapter gives an introduction to the Xilinx IDE from Xilinx, Inc. and the design of the sequential CORDIC architecture and its different components such as the shift register, the LUT etc in the Xilinx IDE.

Fourth chapter gives an introduction to the LISA 2.0 language from CoWare corporation (recently acquired by Synopsis) and some of its typical features and advantages. Secondly, it gives the design of the pipelined cordic architecture using LISA 2.0 language.

Fifth chapter gives the RTL schematics, Device utilization summaries, timing diagrams and page snapshots of the designs implemented using both the Xilinx and CoWare tools.

Sixth chapter gives the conclusion, comparison and future work that could be carried out relating to the CORDIC algorithm based processor design.

CHAPTER 2

Literature Review

Ray,Andraka[5] has carried out a comprehensive research on the different types of CORDIC algorithms for the calculation of various trigonometric, exponential as well as linear functions, transformation to and from polar and rectangular coordinates, the extension to hyperbolic functions in the paper "*A survey of CORDIC algorithms for FPGA based computers".* Besides the survey of the different algorithms, the paper also gives a list of a number of CORDIC processors, such as iterative, bit- iterative and their implementation in an FPGA.

Sung T.Y. , Hsin H.C.[6] in their paper titled *"Design and simulation of reusable IP CORDIC core for special- purpose processors"* have proposed an entirely new extenstion to the in use CORDIC algorithm by the use of double rotation. Such a double rotation increases the efficiency and greatly reduces the efficiency by reducing the time of convergence. It also gives the architecture and implementation of such a CORDIC algorithm on a FPGA.

Reimund Klemm, Javier Prieto Sabugo, Hendrik Ahlendorf, Gerhard Fettweis [7] in their work *"Using LISATek for the Design of an ASIP core including Floating Point Operations"* have provided great insights into the design of ASIPs and how the LISATek tool is helping designers of processors to drastically shorten the design time, at the same time making the designing task easier. Additionally, it integrates IP cores in terms of a floating point processing unit to enable instruction customization. Besides, it gives an analysis of area and delay parameters using different by pass modes and their product.

CHAPTER 3

Design using Xilinx IDE

3.1 Introduction to Xilinx ISE

Xilinx Integrated Software Environment (ISE) [4,8] is a software tool developed by Xilinx corporation for the synthesis and analysis of Hardware Descriptive Language (HDL) designs. It enables the synthesis of designs, timing analysis , Register Transfer Level (RTL) diagram examinations, simulation as per different environments as well as the configuration of the target device with the help of the programmer.

3.1.1 Design fundamentals

The first step in designing of any device in the Xilinx ISE is the creation of a new project with the appropriate information regarding the product category, the family for which the design is being made, the package, the speed grade, language etc. The next step is the creation of a VHDL source file with information regarding the different inputs and outputs to and from the design. Then the new source is created by giving the behavioral description of the design. While doing this various language templates available in the Xilinx library may also be used. Design may also include signals which are connected within the circuit, besides the inputs and outputs.

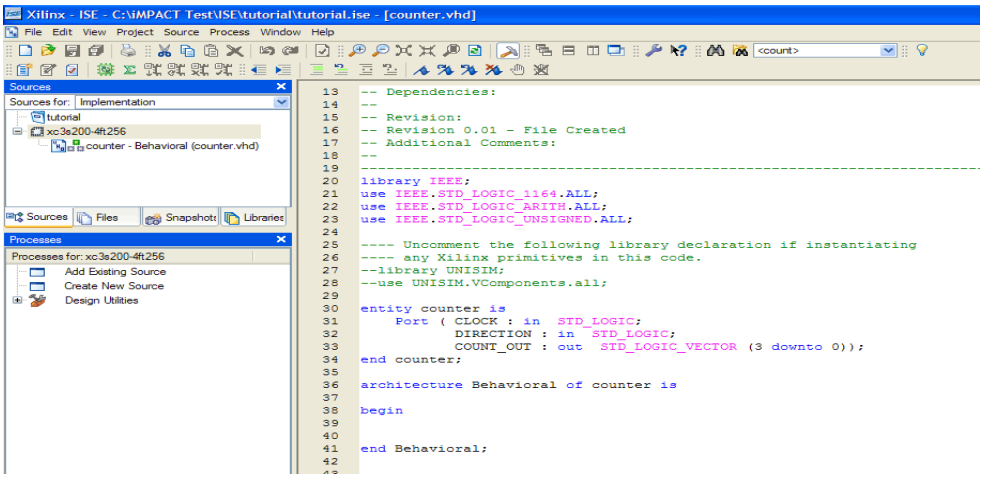


Fig.6. Snapshot of the behavioral description window[4]

3.1.2 Design verification and simulation

After the behavioral description is provided, the syntax is checked for correctness using the syntax process available. After ensuring correctness, the functionality of the design is checked using the behavioral simulation. It involves the creation of a test bench waveform, which is a graphical view of the test bench. Various parameters can be varied in the test bench such as the clock high time, the clock low time, input setup time, output delay, offset etc.

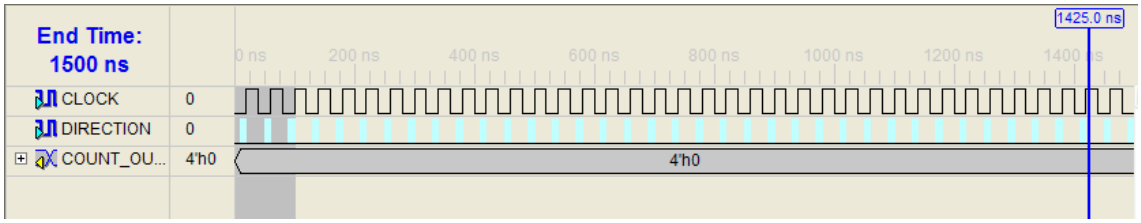


Fig.7. Test Bench Waveform

3.1.3 Implementation of design

After the verification of the design and that of the test bench waveform for proper functioning, the implementation of the device is done. Post – implementation, a summary is generated which contains, besides other things, the device utilization in the design. The Register Transfer Level (RTL) description is generated which gives a clear picture of the hardware components present.

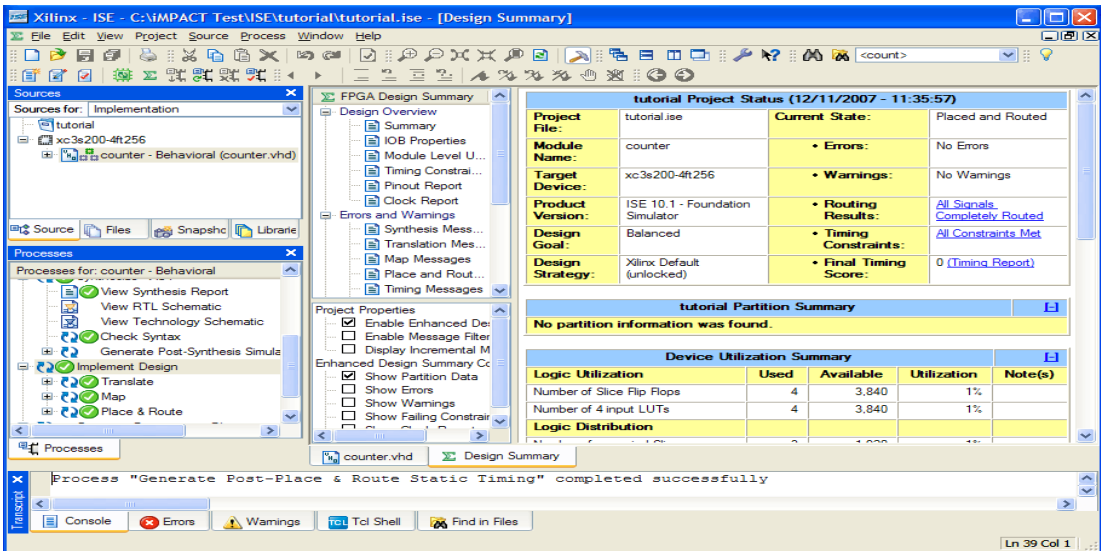


Fig.8. Snapshot of the post-implementation design summary

3.2 Design of CORDIC using Xilinx ISE

The sequential CORDIC algorithm (as shown in fig.2) is implemented using the Xilinx ISE design tool. As seen in the architecture, the main components of the design are the shift registers, the adders and the Look-Up Table (LUT). Structural method of designing is employed in which the different components are implemented separately and then the final structure is made by combination and port mapping of the individual components.

3.2.1 Shift Registers

Shift registers are mainly used for the purpose of storing data. They contain a series of flip-flops connected back to back such that the output from one flip-flop is the input to the next flip-flop. A common clock drives all the flip-flops and they can be set and reset at the same time. Such a shift register is called a *Barrel shift register*. It is mainly a combinational circuit that has n – inputs , n – outputs and other control pins that specify the type and amount of shift. The parameters of shifting specify the direction of shifting

(left or right), the number of bits by which the number is to be shifted and the type of shift operation taking place i.e., circular, arithmetic or logical.

3.2.2 ROM LUT

A Read-Only Memory is essentially a type of memory device in which values are to be pre-stored and can only be fetched / read. However, recent developments in memory technology have bridged the gap between the different types of memory and the data in the ROM devices is easily modifiable. In this case, the ROM is used to implement a LUT which stores the angle values which are to be used in the computation.

A LUT is a type of data-structure, usually an array which is used to replace a run-time computation with a list of stored values. Since retrieving from memory is much faster than computation, it saves time and also reduces hardware complexity which would have been required for the computation. In the current implementation, the arctan values from i=0 to 28 are stored in the LUT, in the 16-point fixed format representation. The obtained angle is compared with the desired angle in each of the iterations which is the deciding factor for the direction of rotation.

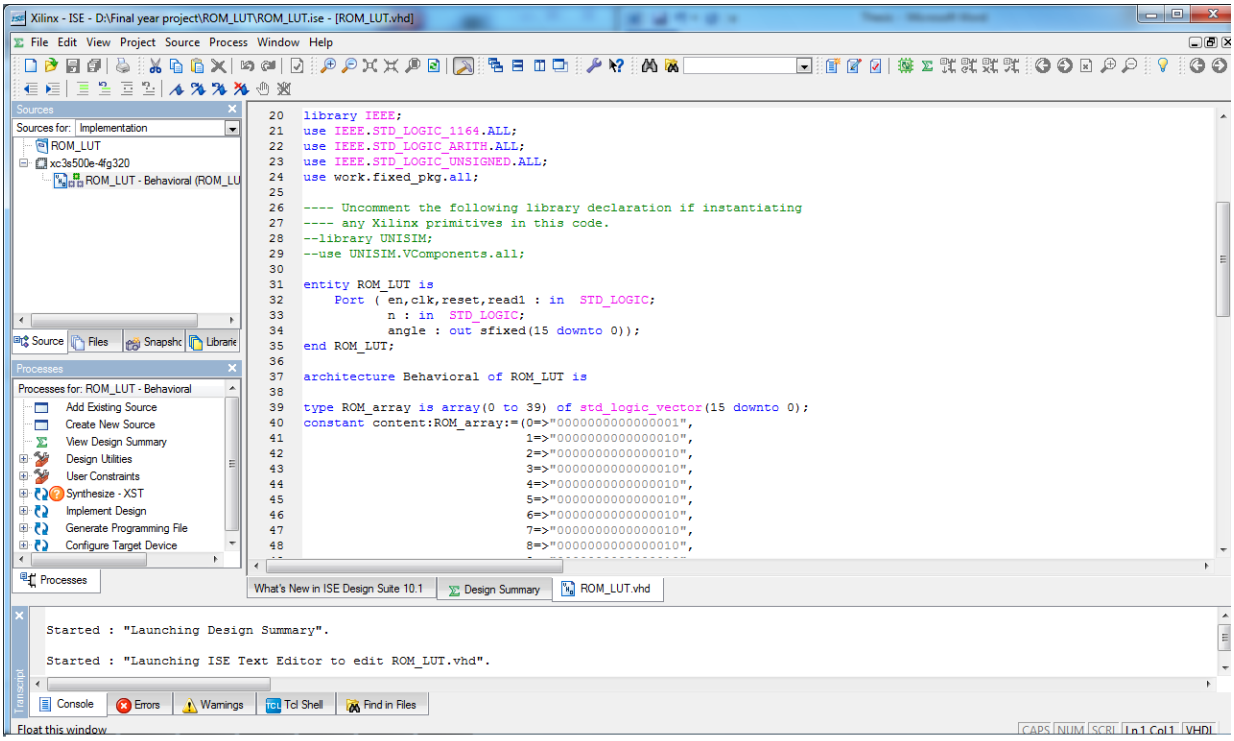


Fig.9. Snapshot of the ROM_LUT behavioral description

3.2.3 Behavioral description

The behavioral description of the model is provided as shown below. The input is the desired angle, for which the sine and cosine values are to be calculated. The i , i.e. the number of iterations that are to be performed is specified depending upon the accuracy of the result desired. Then after the syntax checking, the RTL schematic, test bench waveform and the device utilization summary are generated which are given later in the simulation and results section.

CHAPTER 4

Design using CoWare Processor Designer

4.1 Introduction to CoWare Processor Designer [9]

CoWare processor generato is an automated ASIP design and optimization environment. It was developed by coware corporation and has recently been acquired by synopsis. Some of the major components of the processor designer are the LISA 2.0 language, the processor debugger and the Instruction Set Designer.

4.1.1 Intoduction to LISA

LISA is an acronym which stands for "**L**anguage for **I**nstruction-**S**et **A**rchitectures". It is suited to the modeling of any architecture whose behavior is controlled by an instruction set which is provided in the form of a dedicated resource. The language covers a wide range of architecture types such as General Purpose, RISC , DSP, special-purpose processors besides ASIPs for which it is most suited.

Resources and operations are the main components of a LISA processor model. Resources describe all the storage elements of the processor such as the program memory, data memory, the buses etc while the operations give the transition functions of the processor, including the instructions as well as instruction-dependent functions.

4.1.2 Resource modeling

Processor resources include the storage elements, the input/ output pins, global variables, bus architecture etc. The storage elements include the memoies as well as the registers. Besides these, in cycle- accurate models of which a pipeline is an integral part, other resources such as pipeline registers, interconnect signals. The resources are declared in the resource section, a typical syntax of which is as shown in the next page.

```
RESOURCE
{
    ... RAM char prog_mem
        {
            SIZE(0x1000);
            BLOCKSIZE(8,8);
            FLAGS(R|X);
        }; ...
}
```

The declaration consists of an identifier, a data type specifier and an option to describe the semantic type of the resource.

4.1.3 Modeling instructions

The basic building block of description of an instruction is the OPERATION, which performs a role similar to *classes* in C++. The operation generally consists of three sections such as CODING, SYNTAX and BEHAVIOR.

The OPERATION declaration consists of the *OPERATION* keyword followed by an unique identifier which gives information about the functionality of the operation. Operations may range from *FETCH* and *DECODE* operations, which give the manner in which instructions are obtained from the program counter and how they are executed, to *ADD* and *MUL* operations, which as their name suggest, give the functioning of the arithmetic operations. A typical operation description is as shown :

```
OPERATION identifier
{
        DECLARE { ... }
        CODING { ... }
        SYNTAX { ... }
        BEHAVIOR { ... }
  }
```

Declare section gives the list of resources that the particular operation uses. It gives the instances or the groups of either the immediate operands or the registers and the number of which it is using.

Behavior gives the model of the instruction behavior. It contains the arbitrary C block code which describes the functionality of the operation.

Syntax gives the format in which the assembly instruction is to be provided to the processor for the functioning of the particular operation. It is a usually a string or a sequence of strings.

Coding section gives the binary image of the instruction operands. In the simplest case it gives the opcode, the register which is to be used from the register file and the immediate operand or address if any. It is a sequence of bit-fields, in which the register is specified in 4-bits (out of a register-file of 16 registers), the immediate or address specified in 8-bits and the opcode specified in a number of bits such that the total number of bits is 16.

The tools of the processor generator which require an instruction decoder generation, extract information from the coding sections.

For example, *0b0011 reg16 imm8*

4.1.4 Advantages of CoWare processor designer[9]

- The generation of RTL schematic takes place automatically with both the control as well as the datapaths.
- It also automatically generates the software development tools required i.e. the compiler, the decoder, assembler and linker.
- It is compatible with extensively used tools such as Xilinx , synopsis and cadence.
- It enables flexible designing and greatly reduces design time and engineer-years.

4.1.5 Instruction Set Designer[9]

The ISD is a Graphic User Interface that allows the designer to view, edit as well as create LISA processor models. The option of having a graphical representation rather than just the source code to edit makes processor design much simpler and easier to learn since knowing the details of syntax is not necessary. The ISD and the LISA source code compliment each other and changes made in either is reflected in the other.

4.1.6 CoWare Processor Debugger[9]

The CoWare Processor Debugger is also a Graphic Interface which allows the designer to observe, edit and profile the assembly source code. The GUI shows the current instruction of the assembly syntax which is being executed, the register values and memory values, changes in which are taking place and also the changes in output. It is

intended to analyze and debug the LISA 2.0 processor model. A snapshot of the window is provided below:

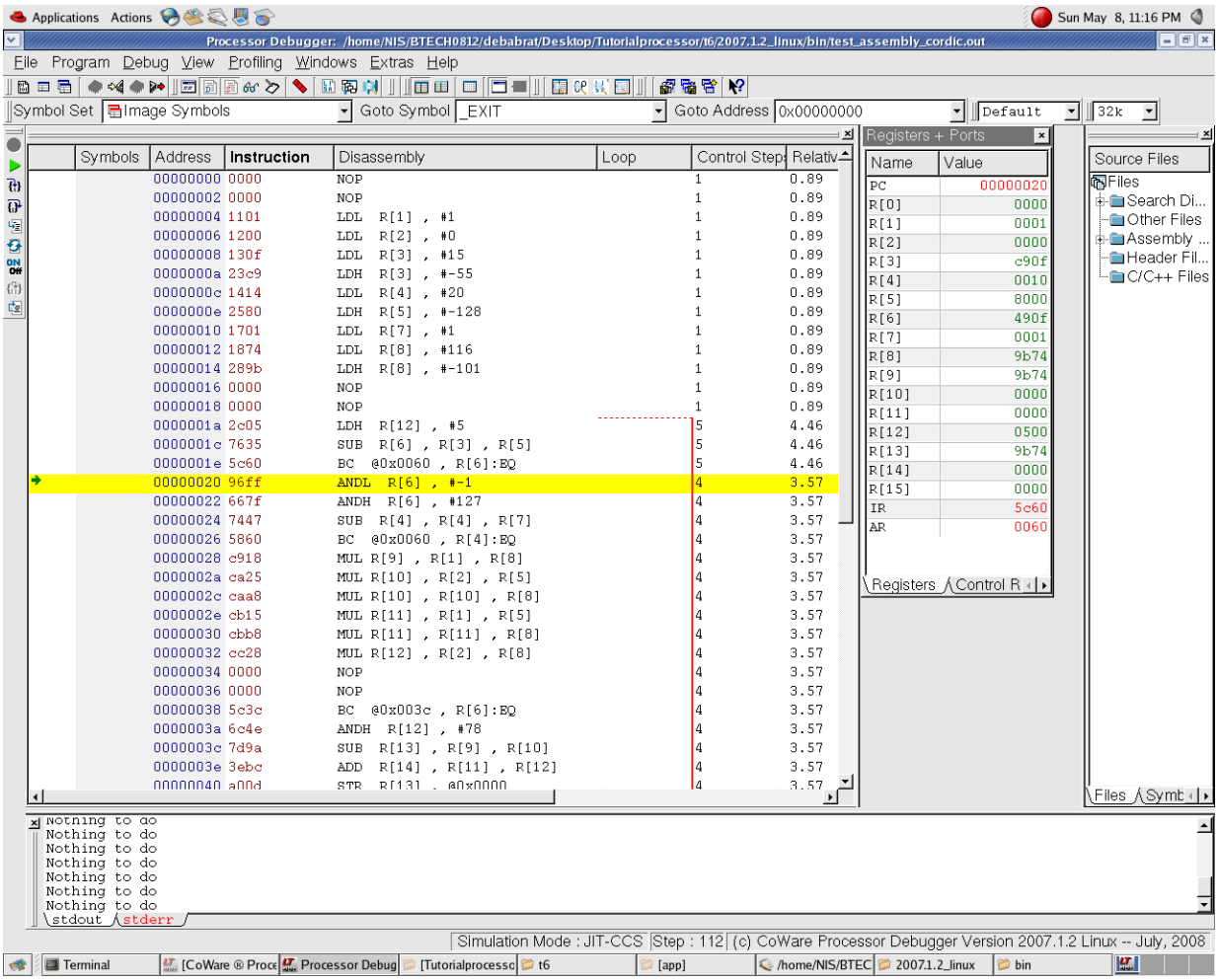
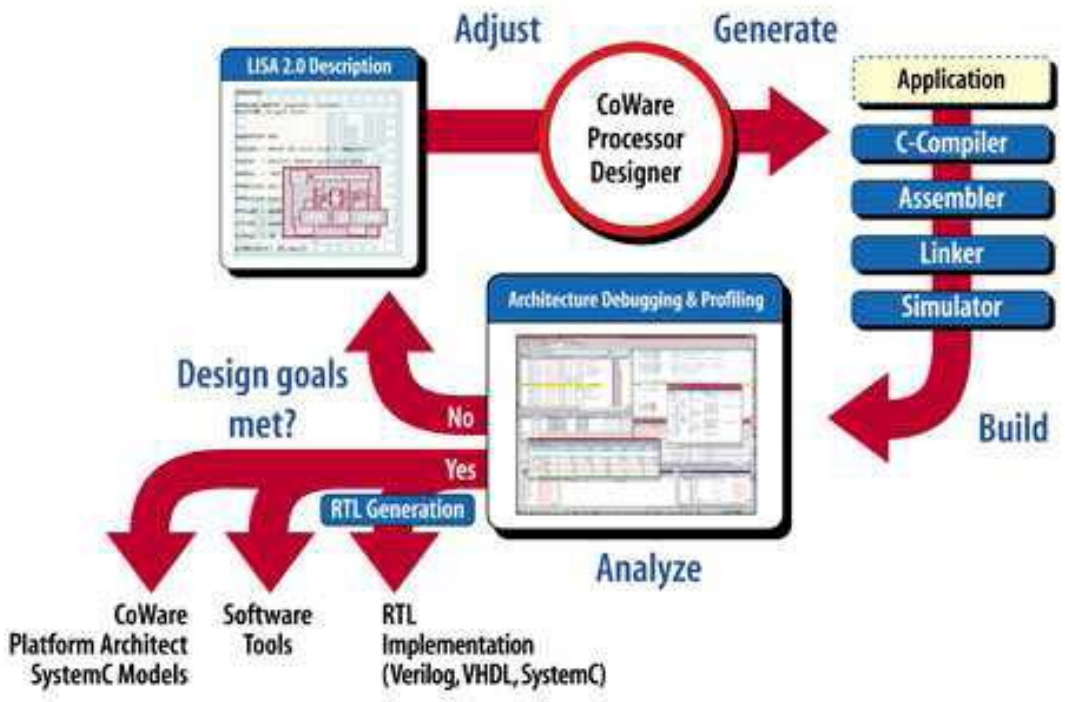


Fig.10. Snapshot of the processor debugger window (with CORDIC assembly code)

4.2 CORDIC Processor design using CoWare

An ASIP based on the CORDIC algorithm as described in the previous sections is designed using the CoWare Design Environment. During this, one of the typical features of CoWare design environment that is taken advantage of is the hardware – software co-design flow. The typical flow of the processor design is as shown below:



[10]

Fig.11. CoWare design flow

The instructions that are required for the implementation of the CORDIC algorithm are estimated from the equations (1) and (2) (refer to chapter 1). The add, subtract and multiply operations are realized by the use of arithmetic operations *ADD, SUB and MUL* whereas the compare instruction is carried out by the subtraction, comparison of MSB and the *BC* (Branch Conditional) operations. The 16-bit register file is used along with the memory-file. A table of the mnemonics, syntax and the operation they perform is given in the following page.

Sl. No.	Mnemonic	Syntax	Operation
1	nop	nop	Performs no operation
2	add	add r1,r2,r3	adds value of GPR r2 to r3 and stores the value in r1
3	sub	sub r1,r2,r3	subtracts value of r3 from r2 and stores value in r1
4	andl	andl rd, #imm8	performs logical and operation between the imm8 value and lower byte of GPR rd
5	andh	andh rd, #imm8	performs logical and operation between the imm8 value and upper byte of GPR rd
6	mul	mul r1,r2,r3	multiplies value of r2 to r3 and stores the value in r1
7	shr	shr r1	shifts the bits of GPR r1 by 1 bit to the right

8	branch conditional, no condition	bc @addr	jumps to location addr specified
9	branch conditional, equal to zero	br @addr, rd:eq	jumps to location addr if value in GPR rd is zero
10	branch conditional, not equal to zero	br @addr, rd:neq	jumps to location addr if value in GPR rd is not zero
11	ldl	ldl rd, #imm8	loads immediate value imm8 to lower byte of GPR rd
12	ldh	ldh rd, #imm8	loads immediate value imm8 to higher byte of GPR rd

4.2.1 Pipelining

Pipelining is one of the most intricate features of the CoWare design environment. Without an instruction pipeline, generation and synthesis of a RTL schematic is not possible in a CoWare design. In the current design, a three stage pipeline is used. The stages are :

- Fetch / Decode (FD)
- Address Generation (AG)
- Execution (EX)

One of the many advantages of pipelining is that, the performance of the processor is significantly increased. This is because although only a single operation of fetch/decode takes place in the first clock cycle, in the subsequent clock cycles multiple operations take place, thus increasing the throughput of the processor.

However the pipelining suffers from many problems known as hazards of pipelining. One such example is that, the same memory locations may be accessed in different pipelining stages most importantly during the address generation and the execution stages. To solve these hazards, pipelining registers are provided and all the fetch/decode instructions are modeled within those registers.

Thus, the CORDIC processor implemented in this case is the pipelined CORDIC architecture.

CHAPTER 5

Simulation and Results

5.1 Simulation results of processor using Xilinx

5.1.1 Synthesis of the ROM LUT

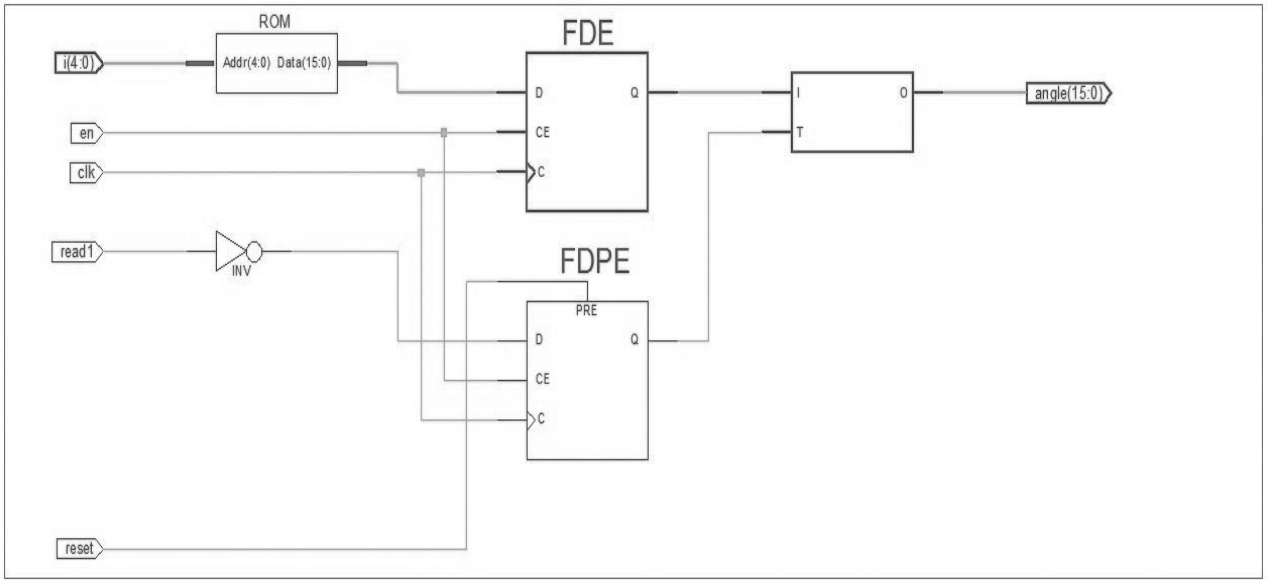


Fig.12. RTL schematic of the ROM LUT

Device utilization

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	1	4656	0%
Number of Slice Flip Flops	18	9312	0%
Number of 4 input LUTs	2	9312	0%
Number of bonded IOBs	21	232	9%
Number of GCLKs	1	24	4%

Table.1. Device utilization summary of the ROM LUT

5.1.2 Synthesis of the Shift Register

Fig.13. RTL schematic of the implemented shift register

Device utilization

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	33	3584	0%
Number of 4 input LUTs	57	7168	0%
Number of bonded IOBs	36	221	16%

Table.2. Device utilization summary of the shift register

5.1.3 Device utilization and test bench waveform of the processor

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	794	3584	22%
Number of 4 input LUTs	1508	7168	21%
Number of bonded IOBs	55	221	24%
Number of GCLKs	1	8	12%

Table.3. Device utilization summary of processor

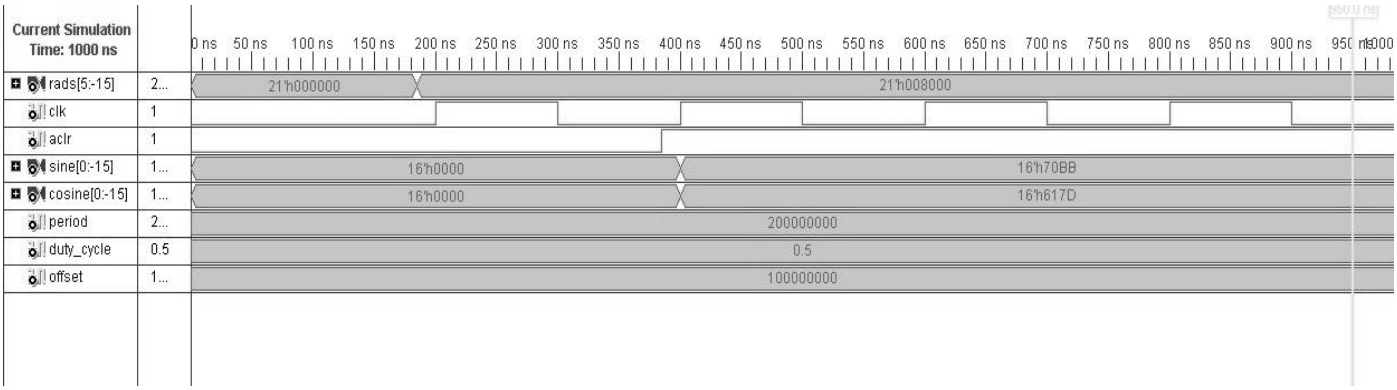


Fig.14. Test bench waveform of the CORDIC processor

5.2 Simulation results of processor using CoWare

5.2.1 Synthesis of the external memory

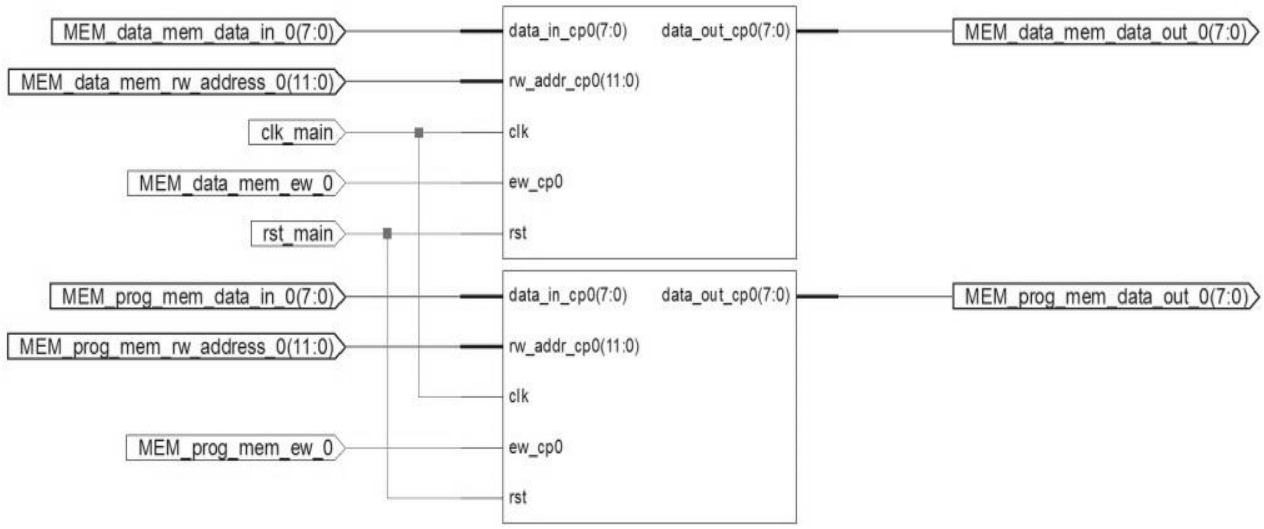


Fig.15. RTL schematic of the external memory showing program and data memories

Device utilization

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	0	4656	0%
Number of bonded IOBs	60	232	25%

Table.5. Device utilization summary of external memory

5.2.2 Synthesis of the processor

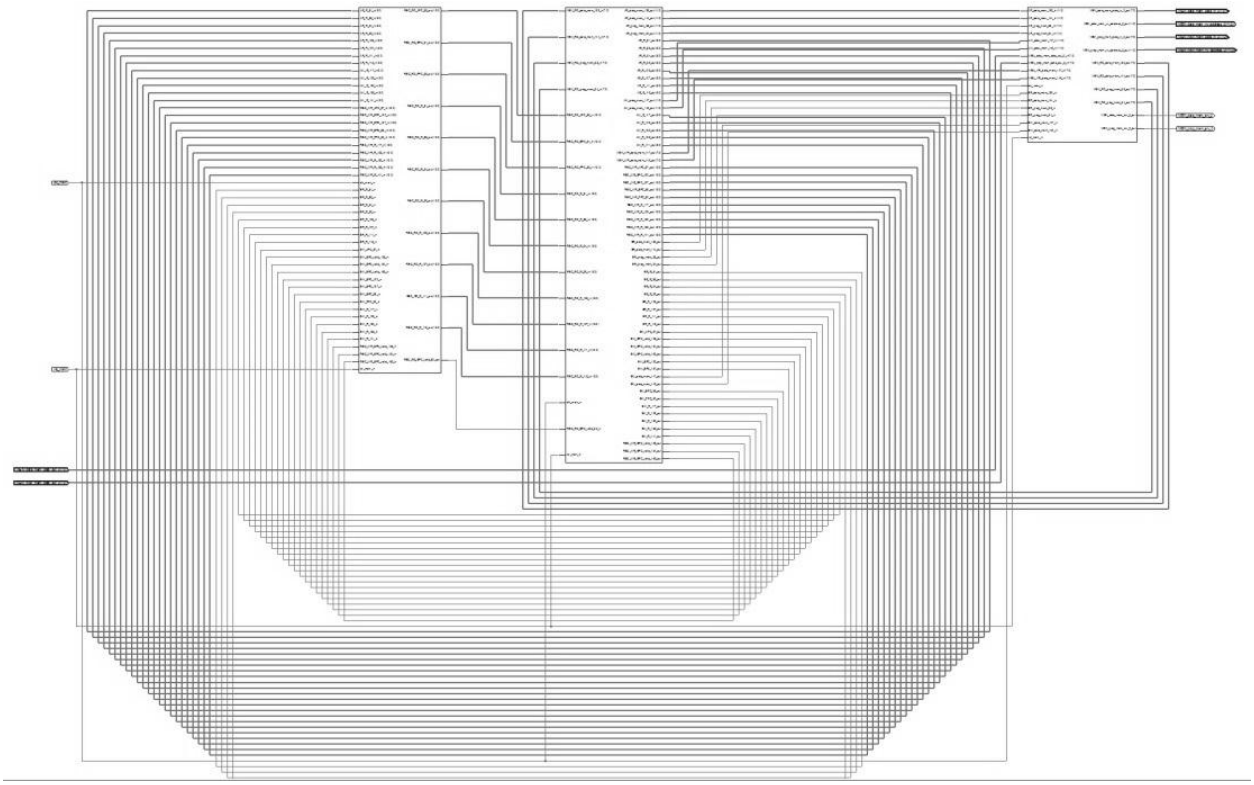


Fig.16. RTL schematic of CORDIC processor in CoWare

Device utilization

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	1661	4656	35%
Number of Slice Flip Flops	378	9312	4%
Number of 4 input LUTs	3197	9312	34%
Number of bonded IOBs	60	232	25%
Number of MULT18X18SIOs	1	20	5%
Number of GCLKs	1	24	4%

Table.6. Device utilization summary of the CORDIC processor in coware

Detailed RTL view

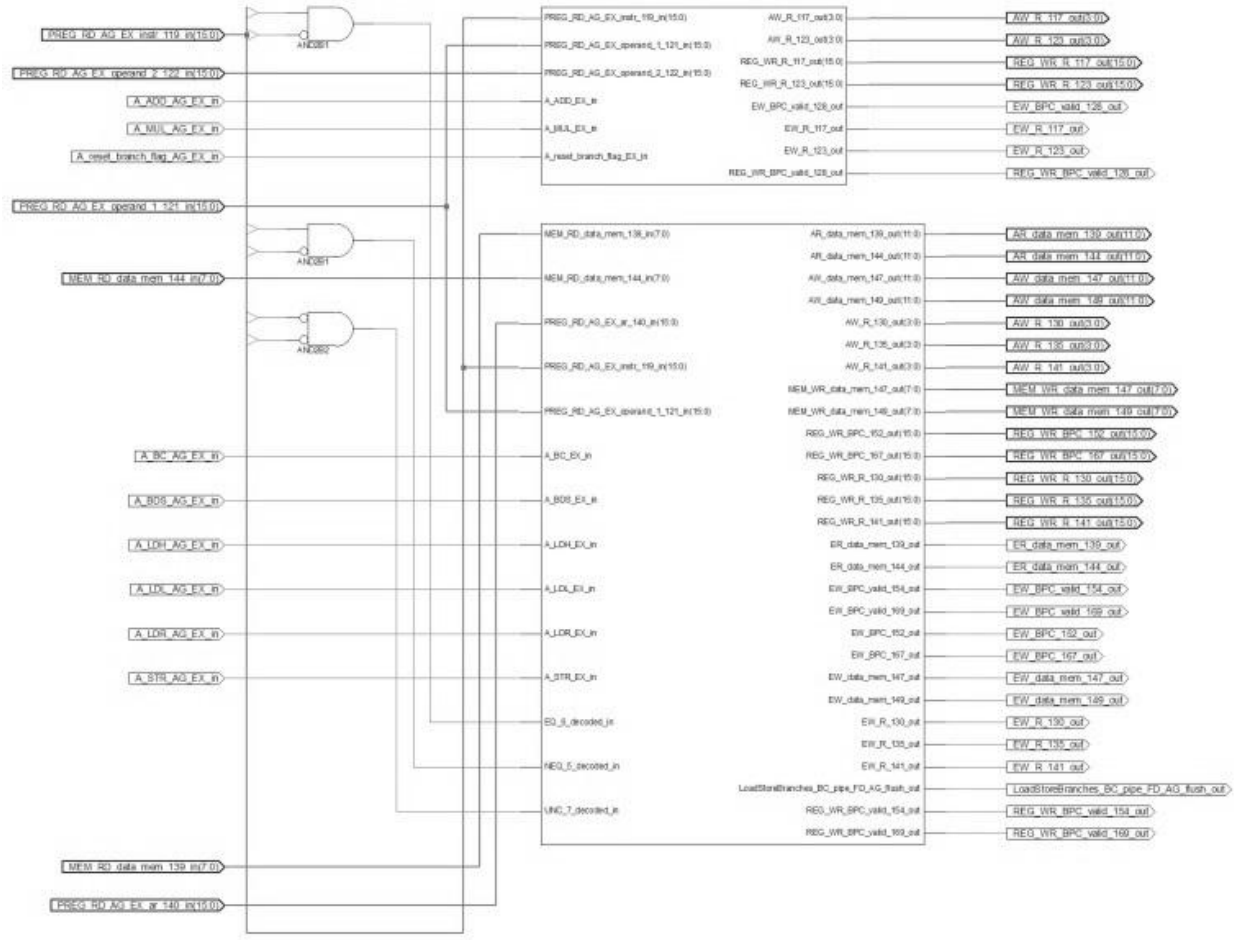


Fig.17. RTL schematic of the arithmetic and load-store branches (level 2)

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	57	4656	1%
Number of 4 input LUTs	103	9312	1%
Number of bonded IOBs	177	232	76%

Table.7. Device utilization summary of fetch/decode operations

CHAPTER 6

Conclusion and Future Work

Conclusion

- The CORDIC algorithm for the calculation of trigonometric and exponential function was reviewed and implemented using both Xilinx IDE as well as CoWare IDE
- The architecture implemented in Xilinx was sequential, which is the simplest of architectures. It has least hardware complexity and ease of design, however it is time-consuming since each new iteration takes place in a new clock cycle.
- The architecture implemented in CoWare LISATek is pipelined, in accordance with the requirements of the processor designer for generation of RTL schematics. The pipelined architecture is more hardware intensive and complex in design, but it has the advantage of being much faster since multiple instructions get executed in the same clock cycle.

Future Work

- FPGA implementation of both the processors designed using Xilinx IDE as well as LISATek could be done.
- Various new and more efficient algorithms of CORDIC rotation could be review and implemented.
- CoWare LISATek is a tool with very powerful and time – saving features, though most of it are a tad difficult to learn. Study and documentation of the software could be done which would be extremely beneficial.

REFERENCES

[1] J.E. Volder, "*The CORDIC Trigonometric Computing Technique*," IRE Transitions on Electronic Computers, vol. EC-8, no.3 , 1959, pp.330 -334

[2] Prof.Kris Gaj, Gaurav Doshi, Hiren Shah, "*Sine/Cosine using Cordic Algorithm"*

[3] Frank Vahid, Tony Givargis, "*Embedded System Design"* pp.9-11

[4] http://www.xilinx.com/publications/products/cpld/logic_handbook.pdf

[5] Ray Andraka,"*A survey of CORDIC algorithms for FPGA based computers"*

[6] Sung T.Y. , Hsin H.C.,"*Design and simulation of reusable IP CORDIC core for special-purpose processors"*

[7] Reimund Klemm, Javier Prieto Sabugo, Hendrik Ahlendorf, Gerhard Fettweis**, "***Using LISATek for the Design of an ASIP core including Floating Point Operations"*

[8] http://en.wikipedia.org/wiki/Xilinx_ISE

[9] Dodani Vicky Rameshlal, Nikhil Kumar, *"EMBEDDED DSP PROCESSOR DESIGN USING COWARE PROCESSOR DESIGNER AND MAGMA LAYOUT TOOL"*

[10] CoWare, "*The LISATek™ Solution-Automated Embedded Processor Design and Software Development Tool Generation"*

[11] *CoWare, The ESL design Leader reference manuals, Product version V2007.1.2,June-08*

[12] http://en.wikipedia.org/wiki/CORDIC

[13] Rashid Muhammad, Ludovic Apvrille, and Renaud Pacalet, "*Evaluation of ASIPs Design with LISATek*"

[14] Oliver Schliebusch, A. Chattopadhyay, R. Leupers, G. Ascheid, H. Meyr, Mario Steinert, Gunnar Braun, Achim Nohl, *"RTL Processor Synthesis for Architecture Exploration and Implementation"*